Remarks

Status of the Application

Applicants respectfully request reconsideration of the rejections set forth in the Office Action mailed on August 12, 2003.

- Claims 1, 3-16, and 18-21 have been rejected under 35 U.S.C. 103 (a)
- Claims 2 and 17 remain cancelled

Thus, claims 1, 3-16, and 18-21 are pending in the current application.

The Claims

Rejections Under 35 U.S.C. § 103

The Examiner has rejected all pending claims under 35 U.S.C. § 103.

Claims 1 and 16

Claim 1 is rejected as being unpatentable over U.S. Patent No. 5,991,173 to *Unger et al.* (*Unger*) in view of U.S. Patent No. 6,163,811 to *Porter* (*Porter*). Applicants respectfully traverse.

Both *Unger* and *Porter* describe various methods of compression of data structures. *Unger* describes methods of compressing natural language text while *Porter* describes token-based source file compression. However, neither Unger nor Porter reasonably suggest or describe the present claim as presented.

Claim 1 explicitly requires, "compressing a portion of compiler information to obtain compressed related compiler information wherein the portion of the compiler information being compressed by said compressing comprises encoded program symbol names...." Thus only a portion of the underlying source code, namely the encoded program symbol names, is compressed. So, for example, in an embodiment of the present claim, a source code file having encoded program symbol names may undergo an initial compression solely of the encoded program symbol names while leaving the rest of the file unaltered. As noted previously, encoded program symbol names are not simply characters in a text file as taught by *Unger*, but are more correctly characterized as programming references or pointers with meaning outside of the

contextual of a text file as taught by *Unger*. Thus, the programming references or pointers under the present claim are initially compressed.

Further, as noted in the Specification at pp. 2-4:

> Source programs typically use a sequence of tokens to name program symbols. The portion 102 illustrated in FIG. 1 declares three program symbols, a "namespace", an "int" variable, and a "long" variable. The "namespace" is a container, and has the two variables as its members. Typically, programs use short context-dependent names for these symbols. The names are "foo", "x", and "y", respectively. Occasionally, programs may use longer, and more complex, context-independent names. They are "::foo", "::foo::x", and "::foo::y", respectively. These names include the names for the symbol's container as well as the identifier for the symbol itself.
>
> Source programs may declare symbols with identical context-dependent names, but different context-independent names. FIG. 2 illustrates another portion 104 of a representative source program written in C++ and serves to declare a "namespace" and two "float" variables. These variables have context-dependent names "x" and "y", which are identical to the "int" and "long" variables declared in FIG. 1. However, their context-independent names are "::babo::x" and "::babo::y", which are different from the context-independent names for the other variables.
>
> An example encoded name for the symbol identified by "x" in FIG. 1 is "__1cDfooBx_", where "__1" is a prefix identifying the particular encoding algorithm, "c" encodes the kind of symbol (function or, in this case, variable), "foo" is the context-dependent name for the container of the symbol, "D" is the length of the string "foo", "x" is the context-dependent identifier for the symbol, "B" is the length of the string "x", and "_" is the name terminator. Likewise, the encoded name for the symbol identified by "x" in Fig. 2 has the encoded name "__1cEbaboBx_".
>
> Unfortunately, however, the encoded symbol names are typically substantially longer than the original context-dependent names. As can be seen in Figs. 1 and 2, even in the simplest cases, the encoding may result in encoded identifiers that are ten times longer than the original context-dependent identifier. Moreover, in more practical applications, symbols can have many levels of containers, with many containers having very complex names. In such cases, the length of the encoded identifiers for symbols can be very long. Encoded identifiers in excess of 5000 characters have been reported in some applications. The length of these identifiers is further compounded when they are used as part of application-specific data for non-critical applications. For example, debuggers typically require more data than is necessary for strict interpretation of the program. This data will use encoded names, which makes the size of the debugging data sensitive to the size of the encoded identifiers. (emphasis added)

One skilled in the art can appreciate that encoded context-dependent identifiers (encoded program symbol names) are often difficult to compress because of their context and uniqueness. Generally speaking, compression schema depends on repeated characters alone and in

combination. Thus, conventional compression techniques (like those described in *Unger* and *Porter*) often leave encoded program symbol names unaltered and uncompressed. As such, neither *Unger* nor *Porter* alone or in combination render the above claim obvious first because *Unger* does not describe compression of encoded program symbol names and because *Porter* does not cure the deficiency in *Unger* because *Porter* describes only compression of entire source files over a distributed network. Furthermore, while Porter describes creating a symbol table for operands (i.e. "=", "+", "-", etc.) and their substitution (*see* col. 6, ll. 17-27; FIG. 3a), *Porter* does not describe encoded program symbol names as contemplated by the present claim.

Therefore, for at least the reasons stated above, Applicants respectfully submit that the rejection of claim 1 is not supported by the cited art and respectfully request reconsideration of the above rejection. Claim 16 is a computer-readable medium claim corresponding to claim 1 and is therefore allowable over the cited art for at least the same reasons stated for claim 1 above.

Claims 3-6, 18, and 19

Claims 3-6, 18, and 19 are rejected as being unpatentable over *Unger* and *Porter* in view of U.S. Patent No. 6,005,503 to *Burrows* (*Burrows*). Applicants respectfully traverse.

Dependent claims 3-6, 18, and 19 depend either directly or indirectly from independent claims 1 and 16 and are therefore also allowable over the cited art for at least the reasons stated for claims 1 and 16. Furthermore, Applicants submit that *Burrows* does not cure the deficiencies in either *Unger* or *Porter*. Still Further, the dependent claims require additional elements that when considered in context of the claimed inventions further patentably distinguish the invention from the cited art. For example, all of the above rejected claims are drawn toward a differential encoding scheme. The Examiner has asserted that a "differential encoding scheme is analogous to the delta encoding" scheme as described by *Burrows*.

Burrows describes delta values as, "each integer to be encoded is the difference between two adjacent integer values in an original input list" (col. 4, ll. 30-34) (emphasis added). Thus, Burrows method depends on determining the difference between two physically proximal values.

In contrast, the present invention determines the difference between encoded program symbol names, which are logically associated. For example:

> FIG. 5A illustrates an exemplary C++ declaration 500 with a namespace "foo" containing a class "bar" containing a function "buz" as would be found in a C++ source program. The encoded names for these symbols are, for example, "__1nDfoo_", "__1nDfooDbar_", and "__1cDfooDbarDbuz6M_v_", respectively.

> The portions "__1", "Dfoo", and "Dbar" are repeated between (as opposed to within) these different encoded names. As a consequence, there is redundancy within the compiler product, and thus the overall compiler product is likewise redundant.
>
> According to the invention, the difference between the encoded container name and the encoded member name is placed into the compiler product instead of the full encoded name for the member. This difference is referred to as the differential name. (Specification p. 10)

Thus, *Burrows* does not describe a methodology that accounts for logical association as contemplated by the present claim. As such, *Burrows* cannot render the above claims obvious in combination with *Unger* and *Porter*.

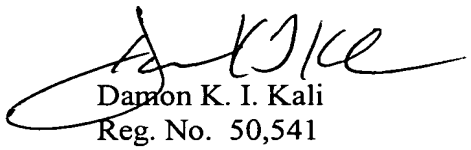
Therefore, for at least the reasons stated above, Applicants respectfully submit that the rejection of claim 3-6, 18, and 19 is not supported by the cited art and respectfully request reconsideration of the above rejection.

All remaining dependent claims depend either directly or indirectly from independent claims 1 and 16 and are therefore also allowable over the cited art for at least the reasons stated for claims 1 and 16. Further, the dependent claims require additional elements that when considered in context of the claimed inventions further patentably distinguish the invention from the cited art.

Applicants believe that all pending claims are allowable and respectfully requests a Notice of Allowance for this application from the Examiner. Should the Examiner believe that a telephone conference would expedite the prosecution of this application, the undersigned can be reached at the telephone number set out below.

Respectfully submitted,

BEYER WEAVER & THOMAS, LLP



Damon K. I. Kali
Reg. No. 50,541

P.O. Box 778
Berkeley, CA 94704-0778
(650) 961-8300